

**REMARKS**

The present amendment is responsive to the Office Action dated March 7, 2006. Claims 1, 8, 10, 12, 13, 18, 20, 22, 23, 46, 48, 50, 51, 55, 57, 59 and 60 have been amended. No new matter has been added by these amendments. Claims 9, 19, 47 and 56 have been cancelled. Therefore, claims 1-8, 10-18, 20-46, 48-55 and 57-63 are again presented for the Examiner's consideration in view of the following remarks.

As an initial matter, claims 8-17 were rejected under 35 U.S.C. § 101 as being drawn to non-statutory subject matter. According to the Office Action, the claimed invention "is non-statutory, since it is not tangibly embodied in a manner so as to be executable, as the only hardware is an intended use statement." (Office Action, pg. 2, numbered paragraph 2.)

According to the Manual of Patent Examining Procedure, "a claimed computer-readable medium encoded with a data structure defines structural and functional interrelationships between the data structure and the computer software and hardware components which permit the data structure's functionality to be realized, and is thus statutory." (M.P.E.P., 8th Ed., Rev. 2, § 2106 at 2100-13, emphasis added.)

Independent claim 8 recites: "A computer readable medium having a software cell recorded therein, said software cell being transmittable over a computer network, said computer network comprising a plurality of processors, said software cell comprising: a program for processing by one or more of said processors; data associated with said program; information for routing said software cell over said network; and a global identification uniquely identifying said software cell among all software cells being transmitted over said network" (emphasis added).

Claim 8 includes a program that can be processed by at least one processor in a network, data associated with the program, routing information for the software cell, and an identifier that uniquely identifies the software cell in the network. The structural and functional interrelationships between the elements of the software cell and the elements of the computer network are clear, and satisfy the requirements for a data structure recorded on a computer-readable medium as outlined above. Thus, claim 8 is statutory. Claims 10-17 depend from independent claim 8 and contain all the limitations thereof. Therefore, applicants respectfully request that the rejection under 35 U.S.C. § 101 of claims 8 and 10-17 be withdrawn.

Claims 1-63 were rejected under 35 U.S.C. § 102(e) as being anticipated by U.S. Patent No. 6,421,736 ("*Breslau*"). Applicants respectfully traverse the rejection. The rejection will be addressed in view of the claims as currently presented.

*Breslau* is directed to an object-oriented computer system that manages instances of objects, in particular migration of objects "between a merged status having a single instance and a split status having multiple instances." (Col. 1, ll. 29-31.)

Independent claims 1, 8, 18, 46 and 55 have been amended to include the limitation of claims 9, 19, 47 and 56, namely wherein the software cell also includes information for routing the software cell over a network.

The computer network of independent claim 1, as currently amended, requires "a plurality of processors connected to said network, each of said processors comprising a plurality of first processing units having the same instruction set architecture and a second processing unit for controlling said first processing units, said first processing units being operable to process software cells transmitted over said

network, each of said software cells comprising a program compatible with said instruction set architecture, data associated with said program, information for routing said software cell over said network, and an identification number uniquely identifying said software cell among all of said software cells being transmitted over said network."

The Office Action asserts that *Breslau* "disclosed a method and system comprising a plurality of processors connected to said network, each of said processors comprising a plurality of first processing units having the same instruction set architecture and a second processing unit for controlling said first processing units (see column 4, line 35 through column 5, line 20)." (Office Action, numbered section 4, pgs. 2-3.) What the cited portion of *Breslau* actually states is:

One example of the hardware elements of a single execution environment computer system used to implement the techniques of the present invention is shown in FIG. 1. A central processing unit ("CPU") 15 provides main processing functionality. A memory 19 is coupled to CPU 15 for providing operational storage of programs and data. Memory 19 may include, for example, random access memory ("RAM") or read only memory ("ROM"). Non-volatile storage of, for example, data files and programs is provided by a storage 13 that may include, for example, disk storage. Both memory 19 and storage 13 are computer usable media that may store computer program products as computer readable program code. User input and output are provided by a user input/output ("I/O") facility 21. User I/O facility 21 may include, for example, a graphical display, a mouse and/or a graphics tablet. System input and output are provided by a system I/O facility 23 (e.g., a network adapter). As will be understood in the art, other I/O facilities may be included based upon system configuration (e.g., between storage 13 and CPU 15). As one computer system example, the computer system of FIG. 1 may be an "IBM APTIVA" Personal Computer System executing an "IBM OS/2 WARP" operating system.

The techniques of the present invention enable the runtime migration of an object between a merged status and a split status. As one example of an object-oriented computer system having an object with a merged status, reference is made to

FIG. 2. Depicted therein is an object-oriented computer system that includes four execution environments, namely three "INTEL" processor-based workstations (workstation A 31, workstation B 33 & workstation C 35) and an "IBM SYSTEM/390" class mainframe-based host computer ("host") 37. The execution environments are interconnected by, e.g., a "TOKEN-RING" network 55 to facilitate communication there between. However, any conventional hardware/software networking scheme may be used in connection with the techniques disclosed herein. Workstations A 31, B 33 and C 35 ("workstations") are designed to interact with users through graphical user terminals 38, 39 & 41 (e.g., each including a monitor, mouse and keyboard), respectively, while host 37 is intended as a database server that accesses stored information on a disk 43.

Both host 37 and the workstations are part of the object-oriented computer system depicted. In regard thereto, a database related object A 53 executes in host 37, and graphical user interface related objects C145, C247 and C349 execute in their respective workstations. An object B 51 performs a function that does not necessarily have to execute on either the workstations or host 37 (i.e., it is not intimately tied to either a disk or graphical interface). For example, object B 51 could perform a numerical computation. In the environment depicted in FIG. 2, object B 51 executes in host 37.

There is simply no discussion in the relied-upon portion of *Breslau* of "each of said processors comprising a plurality of first processing units having the same instruction set architecture and a second processing unit for controlling said first processing units."

The Office Action also contends that *Breslau* teaches the software cell of claim 1, and cites to several sections of *Breslau* in support of this contention. The cited portions of *Breslau* actually state:

Current computer programming techniques include the use of Object-Oriented Programming ("OOP"). Object-Oriented Programming centers on reusable, self-contained, callable program code modules known in the art as "objects." Such a modular approach makes software development more efficient and reliable through the use and reuse of proven, tested objects.

Each object is designed to accomplish a predefined set of operations through "methods." These methods include programmed tasks for the object to perform when called upon to do so (i.e., invoked). Each particular method is defined within a "class" associated with an object. The class acts as a template that describes the behavior of a group of similar objects. An object is instantiated from (or is a runtime instance of) a selected class, and includes methods and attributes (or data) defined in the class.

(Col. 1, ll. 35-50.)

One problem associated with load planning involves object naming. Each object instantiated in a conventional object-oriented computer system is registered in an object manager and identified with a unique name. Invocations of the object are performed through use of the object manager and the object's unique name.

(Col. 1, ll. 61-66.)

As an enhancement, the object may have an identifier assigned thereto for invocation thereof. This identifier is maintained for invoking the object, despite the object having the split or merged status. Further, the object-oriented computer system may include a routing table used by an object manager to track objects. If the object is migrated from the split status to the merged status, then the single instance is assigned to the identifier within the routing table. If the object is migrated from the merged status to the split status, then the at least two instances are assigned to the identifier within the routing table.

(Col. 2, ll. 38-48.)

The cited portions of *Breslau* simply do not teach that each software cell comprises a program that is compatible with the instruction set architecture of each of the first processing units, that data is associated with the program of each software cell, or that the identification number uniquely identifies the software cell among all of said software cells that are transmitted over the network.

As mentioned above, claim 1 has been amended to require information for routing the software cell over the network. With regard to canceled claims 9 and 19, the Office

Action asserted that *Breslau* discloses "wherein each said software cell further comprises information for routing said software cell over said network (see column 6, lines 4-20; column 8, lines 4-20)." (Office Action, numbered section 12, pg. 5) With regard to canceled claims 47 and 56, the Office Action asserted that *Breslau* discloses "wherein each said software cell further comprises information for routing said software cell over the computer network (see column 8, lines 15-54)." Applicants respectfully disagree. The first cited portion of *Breslau* states:

The process begins with the generation and/or receipt of a "split" request (STEP 61). This may be generated by another program or object executing within the object-oriented computer system. As one example, the split request may be generated by a performance monitoring program that has determined an overload condition for a "merged" object within an execution environment. For example, with respect to the example object-oriented computer system of FIGS. 2-3, a performance monitor may detect that the single instance, object B 51 is overloaded on host 37, and accordingly generates a "split" request therefor, to predetermined execution environments determined by the heuristics of the performance monitor. A parameter list is passed from the performance monitor to the current process that enumerates the locations where the split instances are to be placed. The parameter list also indicates whether the original "merged" instance is kept intact.

(Col. 6, ll. 4-20.)

The identified process is "for migrating an object from a merged status (i.e., having a single instance) to a split status (i.e., having multiple instances)." (Col. 5, ll. 63-66.) The second cited portion of *Breslau* states:

ORB table management, according to the techniques of the present invention, is discussed below. Conventionally, an ORB Routing Table tracks objects registered within the object-oriented computer system. Included in this table is an identifier for each object, along with pertinent information associated therewith. For example, the name of the particular execution environment the

object is loaded within is stored in the table along with each object's identifier. According to the techniques of the present invention, modification to the ORB routing table is performed to support objects having split and merged status.

Shown in FIG. 6 is an example of an ORB Routing Table corresponding to the computer system configuration depicted in FIG. 2. Each entry within the table includes an object identifier ("OBJECT ID") and the execution environment that the corresponding object instance is loaded within ("LOCATION"). By way of example, each object instance shown in FIG. 2 is represented in the ORB Routing Table of FIG. 6. In particular, object A 53 is within host 37, object C145 is within workstation A 31, object C247 is within workstation B 33, object C349 is within workstation C 35 and object B 51 is within host 37. A "splittable" field is included for each object identifier and indicates whether the object can be split using the techniques disclosed herein. The "splittable" indicator is set by the object as part of the registration with the ORB. For example, it may be passed as a parameter to the ORB registration function. Also, a Split Instance Pointer ("SIP") points to a secondary routing table for split instances. If the object is not split, a null is stored in the SIP.

The ORB Routing Table of FIG. 7 corresponds to the system configuration of FIG. 3, wherein "object B" has been split into multiple instances thereof, namely objects B161, B263 and B365. The techniques of the present invention track these multiple instances using a "Split Routing Table" for each split object. As shown in FIG. 7, the object identifier "Object\_B" is associated with a SIP corresponding to a Split Routing Table for Object B. Each entry within this Split Routing Table contains the name of each "object B" instance, along with the name of the execution environment that the instance is within. Specifically, object B161 is within workstation A 31, object B263 is within workstation B 33 and object B365 is within workstation C 35. Thus, using these tables in connection with conventional ORB functionality, the identification, tracking and instantiation of the "split" instances of an object, e.g., "object B," are facilitated. If the merged instance of object B were maintained, the identifier "object B" would be maintained in the ORB routing table location field denoting a "partially split" state of object B.

(Col. 8, ll. 4-54.)

According to *Breslau*, "ORB" stands for Object Request Broker, "which is responsible for, e.g., creation, deletion and tracking of objects." (Col. 4, ll. 13-14.) While the ORB Routing Table may "track objects registered within the object-oriented computer system," the cited sections of *Breslau* simply do not teach that the software cell includes information for routing the software cell over the network.

Thus, for at least these reasons, applicants submit that *Breslau* does not teach all of the elements of claim 1. Claims 2-7 depend from independent claim 1, and contain all the limitations thereof. Applicants respectfully request, therefore, that the rejection of claims 1-7 be withdrawn.

Independent claim 8 recites "A computer readable medium having a software cell recorded therein, said software cell being transmittable over a computer network, said computer network comprising a plurality of processors, said software cell comprising: a program for processing by one or more of said processors; data associated with said program; information for routing said software cell over said network; and a global identification uniquely identifying said software cell among all software cells being transmitted over said network."

Independent claim 18 recites "A computer system, comprising: a computer network comprising a plurality of processors; and a plurality of software cells configured for transmission over the computer network, each of the software cells comprising: a program for processing by one or more of said processors; data associated with said program; information for routing said software cell over said network; and a global identification uniquely identifying said software cell among all software cells being transmitted over said network."

Independent claim 46 recites "A computer system, comprising: at least one processor configured for receiving and processing software cells transmitted over a computer network; each of the software cells comprising: a program for processing



by said at least one processor; data associated with said program; information for routing said software cell over the computer network; and a global identification uniquely identifying said software cell among all software cells being transmitted over the network."

Independent claim 55 recites "A processor, comprising: a processing element including a bus, a processing unit and at least one sub-processing unit connected to said processing unit by the bus; wherein at least one of said processing unit and said at least one sub-processing unit are configured to process software cells received from a computer network, each of the software cells comprising: a program for processing by one of said processing unit and said at least one sub-processing unit; data associated with said program; information for routing said software cell over the computer network; and a global identification uniquely identifying said software cell among all software cells being transmitted over the network."

Claims 8, 18, 46 and 55 each include a software cell that has a program, data associated with the program, network routing information and a unique global identification. For the reasons explained above, *Breslau* does not teach that such information is included in a "software cell" as claimed. *Breslau*, therefore, fails to teach all of the limitations of independent claims 8, 18, 46 and 55. Claims 10-17, 20-27, 48-54 and 57-63 depend from independent claims 8, 18, 46 and 55, respectively, and contain all the limitations thereof. Applicants respectfully request, therefore, that the rejection of these claims in view of *Breslau* be withdrawn.

Furthermore, independent claim 55 requires that the processor have a specific structure. The Office Action does not address the claimed processor structure at all in rejecting the claim. "Regarding claims 46 and 55, *Breslau* disclosed a method and system comprising at least one processor configured for receiving and processing software cells transmitted over a

computer network (see column 4, lines 58-67)..." (Office Action, numbered section 39, pg. 10, emphasis added.) This is not what is claimed. Rather, claim 55 is for a processor having "a processing element including a bus, a processing unit and at least one sub-processing unit connected to said processing unit by the bus." The cited section of *Breslau* states:

The techniques of the present invention enable the runtime migration of an object between a merged status and a split status. As one example of an object-oriented computer system having an object with a merged status, reference is made to FIG. 2. Depicted therein is an object-oriented computer system that includes four execution environments, namely three "INTEL" processor-based workstations (workstation A 31, workstation B 33 & workstation C 35) and an "IBM SYSTEM/390" class mainframe-based host computer ("host") 37.

(Col. 4, ll. 58-67.)

This section of *Breslau* simply does not teach the structure as claimed in independent claim 55. Thus, applicants respectfully submit that the rejection of claim 55 be withdrawn. Claims 57-63 depend from independent claim 55 and contain all the limitations thereof. Applicants respectfully request, therefore, that the rejection of the subject dependent claims also be withdrawn.

With regard to independent claim 28, the Office Action contends that "*Breslau* disclosed a method and system comprising storing in said main memory said programs and said data associated with said programs (see column 6, lines 28-45); directing with said second processing unit any one any one of said first processing units to process one of said programs (see column 5, lines 30-40); directing with said second processing unit said memory controller to transfer said one program and data associated with said one program from said main memory to the local memory exclusively associated with said one first processing unit (see column 6, lines 4-20); instructing with said second processing unit said one first processing unit to

initiate processing of said one program from said one first processing unit's local memory (see column 8, lines 59-67); and in response to said instructing, processing with said one first processing unit said one program and said data associated with said one program from said local memory exclusively associated with said one first processing unit (see column 9, lines 9-21). What the cited portions of *Breslau* actually state is:

Optionally, the state information accumulated within the object being "split" may be acquired (STEP 65). As known in the art, as an object is repeatedly invoked, certain variables within the object may persist between invocations such that the information contained thereby is accumulated. For example, an object may include an invocation persistent variable that tracks the total number of times an object has been invoked. This accumulated information is known as the "state" of the object and may be important to methods of the object. Thus, saving the "state" of the object facilitates preservation and reimpression thereof into the "split" instances of the object once created. The saving and reimpression of state information are described in detail in "METHOD AND SYSTEM OF DYNAMICALLY MOVING OBJECTS BETWEEN HETEROGENEOUS EXECUTION ENVIRONMENTS," Ser. No. 08/578,098, filed herewith on Dec. 27, 1995, which is hereby incorporated herein by reference in its entirety.

(Col. 6, ll. 28-46)

During runtime system operation, it may become desirable (or even necessary) to increase the processing capacity of object B 51. For example, the load on object B 51 may become too great for host 37 to service properly. According to the techniques of the present invention, the single instance, object B 51 is "split" into three instances thereof depicted in FIG. 3 (and in this example, thereafter deleted). For example, the three split instances shown are an object B161 within workstation A 31, an object B263 within workstation B 33 and an object B365 within workstation C 35. Each of these instances are instantiated from the same class (or classes) and are therefore functionally identical.

(Col. 5, ll. 30-40)

The process begins with the generation and/or receipt of a "split" request (STEP 61). This may

be generated by another program or object executing within the object-oriented computer system. As one example, the split request may be generated by a performance monitoring program that has determined an overload condition for a "merged" object within an execution environment. For example, with respect to the example object-oriented computer system of FIGS. 2-3, a performance monitor may detect that the single instance, object B 51 is overloaded on host 37, and accordingly generates a "split" request therefor, to predetermined execution environments determined by the heuristics of the performance monitor. A parameter list is passed from the performance monitor to the current process that enumerates the locations where the split instances are to be placed. The parameter list also indicates whether the original "merged" instance is kept intact.

(Col. 6, ll. 4-20)

When the ORB receives an invocation request for an object that has a split status, the ORB must decide which of the multiple instances of the object to invoke. The heuristic used to make such a decision will vary by object purpose and application. One example of a usable heuristic would be, e.g., a round-robin selection scheme based upon the list of instances within the Split Routing Table. Other heuristics may use information regarding the execution environment on which the invocation originated (conventionally available).

(Col. 8, line 59 to col. 9, line 1)

A further feature of the present invention includes informing an object, upon invocation, whether it has a split or a merged status. If an object is merged, and therefore contains only one instance, a conventional invocation of it would inherently inform the object that it exists as a merged object. If the object has a split status, the ORB may pass the invoked instance of the object a dedicated parameter upon invocation to indicate its "split" status. The ORB will know when it handles the invocation of an instance of a split object, because reference to a Split Routing Table is required to process the invocation. The invoked object instance may then use this knowledge of its split/merged status within its methods.

(Col. 9, ll. 9-21)

The cited portions of *Breslau* simply do not teach all of the limitations of independent claim 28. For instance, while an object may be split into multiple instances that may be run on different workstations, *Breslau* does not disclose directing with a second processing unit any one of a plurality of first processing units to process a programs, when the second processing unit and the plurality of first processing units are part of a computer processor. *Breslau* also does not disclose "directing with said second processing unit said memory controller to transfer said one program and data associated with said one program from said main memory to the local memory exclusively associated with said one first processing unit." *Breslau* does not teach or otherwise suggest exclusively associating a local memory with a "first processing unit" and, for at least the reasons stated above, does not disclose a plurality of first processing units in the configuration presented in claim 28. While the ORB may decide which of the multiple instances of a split status object to invoke, *Breslau* does not teach "instructing with said second processing unit said one first processing unit to initiate processing of said one program from said one first processing unit's local memory" as claimed. Furthermore, because *Breslau* does not teach or suggest a local memory exclusively associated with a first processing unit, it cannot teach "processing with said one first processing unit said one program and said data associated with said one program from said local memory exclusively associated with said one first processing unit."


Thus, for at least these reasons, *Breslau* does not anticipate independent claim 28. Claims 29-45 depend from independent claim 28 and contain all the limitations thereof. Applicants respectfully request, therefore, that the rejection of claims 28-45 be withdrawn.

As it is believed that all of the rejections set forth in the Office Action have been fully met, favorable reconsideration and allowance are earnestly solicited.

If, however, for any reason the Examiner does not believe that such action can be taken at this time, it is respectfully requested that he telephone applicants' attorney at (908) 654-5000 in order to overcome any additional objections which he might have. If there are any additional charges in connection with this requested amendment, the Examiner is authorized to charge Deposit Account No. 12-1095 therefor.

Dated: June 1, 2006

Respectfully submitted,

By   
Andrew T. Zidel  
Registration No.: 45,256  
LERNER, DAVID, LITTENBERG,  
KRUMHOLZ & MENTLIK, LLP  
600 South Avenue West  
Westfield, New Jersey 07090  
(908) 654-5000  
Attorney for Applicant